About the Project

The submitted game is a procedurally generated platformer (Online, Stochastic). The player plays as a bean and needs to reach the goal by jumping over walls, avoiding/killing enemies, jumping over gaps in the platform. A fitness test is used to make sure that the generated level fits difficulty criteria otherwise another level is generated (a very basic version is implemented and needs further exploration). Further, the game learns from player data and adapts difficulty to provide optimal experience (a very basic version is implemented and needs further exploration)

Disclaimer: The assets and animations are taken from Unity Asset Store. Some basic code attached to these animations is also taken and modified. Most of the code is completely rewritten.

Motivation

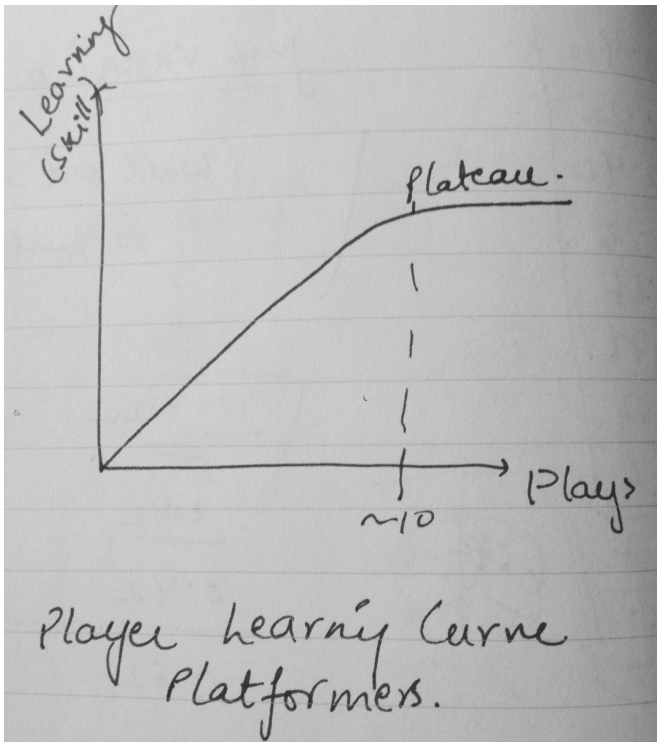
Several psychological theories like Self Determination Theory [1], Flow Theory [2] have proven that people find maximum satisfaction or feel most engaged when they are performing a task they are interested in and the task challenge is just beyond their reach. According to these theories people like to get better at what they do by taking up tasks slightly more challenging than previous. Although if a task becomes to hard or out of reach they lose engagement. Nick Lazzaro[3] in her 4 keys to fun also claims that the above is valid in a video game context, however she has not concretely proven it. Rigby and Ryan[4] have done several studies to prove the Self Determination Theory in games but there is a lack of focus in proving each of their components.

My project is an aim to study if in a procedurally generated world (casual game world as my PhD focuses on casual games), the game can learn and adjust difficulty based on the above theories and later on see how it affects player's immersion and engagement by asking them to fill a questionnaire. The chosen topic for the project was competence. According to Self Determination Theory one of the 3 factors guiding engagement in people is the fact that they feel competent at a task described above.

Design

Choice of Game

A platformer is chosen as it has a known player experience learning curve (from my previous studies in the industry)



and has limited player controls to minimise bias because of unintuitive controls. (I could not completely achieve this in the submitted Unity project although it is in the right direction and can be easily improved).

AI Design and Behaviour

The AI was designed completely keeping player experience in mind and such that it could be playtested and improved in the limited time of project development. The aim was to see if the PCG environment can adapt itself to suit player needs. Via the playtesting, it was learned that in majority of the cases this was achieved (needs statistical experimentation to support this claim).

PCG Design

The PCG is Online, Stochastic and has both a constructive generation but in a different mode it is ‘Generate and Test’ (I should have made two separate projects to clearly show this in code but current submission will need removing of some data for the difference to be seen)

The game has two modes.

In Design mode players can adjust the challenge level and see the change in the generated level. It should be a helpful tool for designers to change a level and test with players to see what is the right for the majority of players who fall in the target audience demographic and create the right level difficulty curve.

Generation Logic

Inspired by Shaker et al. Mario level generation described in the PCG book [5].

The components being generated are -

- The walls with dynamic heights
- Platform gaps with dynamic gap width
- Enemy with dynamic spawn positions and spawn frequency
- Level length (made constant in the submitted project for playtest reasons)

All 3 pick noise from a curve (other methods were tried: Perlin gives a clustered value). On x axis is a random number and on y axis the competence value.

The level is always generated in positive x direction in the below sequence

- A random level length is decided
- Platforms are spawned with gaps using the noise curve
- Bricks are spawned such that they don't spawn inside the gaps, bric height is decided based on its noise curve
- Enemies are spawned within the player range based on its noise curve and the same curve decided it's frequency.
- Backgrounds are spawned dynamically as the player run towards the goal.
- On changing value on the challenge scale all of the above are dynamically generated again.

Fitness Test (Generate and Test Method)

In this method, instead of getting a competence value from the curve a random value is taken.

A fitness function then sees if the generated level has the right difficulty for the randomly unrelated competence value on the scale. If the two are not a match, a new level is generated.

To keep this simple understanding the scope of the project, the competence values were bucketed in 3 categories : Low, medium and High.

Difficulty here is a function of total gap width, total brick height and enemy spawn frequency. In low competence we needed the gap width to be low, brick height to be low and spawn frequency to be high. Currently we just add the first two and subtract the third. That SIMPLE! And the generated 'd' is compared with the generated bucket of competence and then a decision is made to generate the level or not.

(In code, you will have to remove the noise curve multiplier to see this work better. That value is there for the constructive method used for faster playtest for the next section of the design, as a next step I will smoothen to code to make it more generic).

Learning

Since we know the player learning curve of a platformer [mentioned above], the project was designed to modify the levels for the players to smoothly have an optimal experience within 7 re-runs.

For each player their data is stored in a json file. The data stored is

- Win or Loss (whether they reached the goal or not)
- Time taken before the game ended (whether they reached the goal or died)
- Distance travelled before the game ended.

Now depending on the attempt number we know whether how much a player improves as he explores controls, gets used to environment etc.

The first level is randomly generated as we don't know anything about the player yet

An simple algorithm is designed to see how soon should the player start winning, how close should he get to the goal and the competence is adjusted.

For example if the player wins the game in the first attempt the game is too easy but if he struggles to win i.e. takes a long time to reach the goal then although the level is easy as he made it in first attempt it is not that easy.

The game gets tougher with each attempt but then adjust based on player's previous performance.

This whole algorithm is rather handcrafted for now and needs improvement but within the scope of the project the idea was to learn via playtesting of the approach of the AI is in the right direction. The program was tested for several attempts - we know a casual gamer takes about 10 attempts to get to a place where they feel at ease with the environment and controls we stuck around that number.

This algorithm can easily be made generic and improved from here.

Improvements

- Both the fitness function and learning need to be made better and scalable. Gpt really sick during the last 2 days of the project else would have worked on this.
- Player should know how far he is from goal to give them more satisfaction during play (UI element)
- Take player name to remember their performance for future.

- Overall polish and then run some experiments.

References

1. Deci, Edward L., and Richard M. Ryan. "Self-determination theory." *Handbook of theories of social psychology* 1 (2011): 416-433.
2. Csikszentmihalyi, Mihaly. "Flow and the psychology of discovery and invention." *New Yprk: Harper Collins* (1996).
3. Lazzaro, Nicole. "Why we play games: Four keys to more emotion without story." (2004).
4. Przybylski, Andrew K., C. Scott Rigby, and Richard M. Ryan. "A motivational model of video game engagement." *Review of general psychology* 14.2 (2010): 154.
5. Section 5.6.2. ‘Grammatical evolution level generator’, pcgbook.
   Shaker, N., Nicolau, M., Yannakakis, G.N., Togelius, J., O’Neill, M.: Evolving levels for Super Mario Bros. using grammatical evolution. In: Proceedings of the IEEE Conference on Computational Intelligence and Games, pp. 304–311 (2012)